Course conclusion: The Big Picture
Course, Fall 2005

Henning Niss

IT University of Copenhagen

Lecture 12
Before the course

Most applications developed for *single user, single machine*. 
- only one instance ("the user");
- application logic and GUI in one program;
- little shared data;
- …
Example: A Joke Collection

A program maintaining a (local) collection of jokes would be an example of this.

- jokes stored in local file;
- user interface for telling a joke;
- user interface for querying the local joke collection.

Obvious variation:

- use a (remotely available) database;
- (but you didn’t write that : - ) )
Web applications and services

Course definition of a web service/web application:

A web service (web application) is dynamic functionality available over the HTTP protocol.

Analyzing this:

▶ "dynamic functionality": more than just static pages
  ▶ ranging from personalized pages,
  ▶ over form-based interaction,
  ▶ to method calls.

▶ "HTTP protocol":
  ▶ accessible from browsers;
  ▶ remote invocation;
  ▶ don’t worry about transport.
Web applications: “Personalized pages” and web interfaces

At the presentation end of the spectrum:

- customized pages handling input;
- Java Server Pages well-suited
  - (intuitively: HTML pages with embedded code).

Many similar technologies:

- ASP (Active Server Pages);
- xSP (language x Server Pages);
- PHP (Personal Home Page/PHP Hypertext Preprocessor).
Example: Joke Client

Something like the Joke Client would end up in this category:

- client would show the *user interface* to the system;
- mostly a matter of accepting input and constructing parameters;
- ...

Many other examples:

- just about any ASP or PHP page out there.

Key API routines:

- the `HttpServletRequest.getParameter` method;
- (XML processing).
Why Java?

Why Java:

- well-known language;
- nice language properties
  - well-defined semantics;
  - security properties (garbage collection, sandboxing, runtime checks);
  - Unicode;
  - standard libraries (network, XML).
- good support;
- available everywhere.
Why XML?

Why XML:

- de-facto standard for information exchange;
- platform and language independent;
- good tool support;
- key web service technologies XML based.
Form processing

The Servlet API specification defines a standard for handling form processing in Java:

- JSP access;
- Servlet access
  - (intuitively: code with embedded HTML).
- Servlets typically well-suited for more processing-oriented tasks.

Again other similar technologies:

- CGI scripts.
Example: Joke Server

The Joke Server would likely end up in this category:

- mainly a matter of maintaining a collection of jokes;
- "processing-oriented";
- no user interface;

Key API routines:

- the HttpServletResponse.getWriter method.
Servlets vs JSP

Really just two sides of the same story:

▶ seen how to translate a JSP page into a Servlet;
▶ in principle possible to translate in the other direction as well;
▶ difference mostly a matter of perspective (and application).

The translation:

▶ HTML/XML → out.write("...");
▶ <% expression %> → out.print(expression);
▶ <% statement %> → statement
▶ <%! declaration %> → declaration (in Servlet class)
▶ <%=@ directive %> → instruction to translator, e.g. include file

Roughly what Tomcat does to JSP files.
Web application structures

Web applications: really client/server structures.
More language support

Servlets: Some key issues dealt with very indirectly:
- generation of HTML;
- sessions;
- validation.

JWIG & Xact: “languages” incorporating these directly.
- first-class XML values;
- sessions as sequential code;
- static analysis of code;
- validation of form data.
Example: Joke project in JWIG / Xact

Most of the code of the Joke project would be the same, but other guarantees:

- **Joke server:**
  - valid XML responses;

- **Joke client:**
  - validated form data;
  - (in principle: would know that all parameters are supplied);
  - more explicit session structure (code change).

Key language features:

- XML templates;
- session as sequential code;
- static analysis of generated XML and form interaction.
public class Test extends HttpServlet {
    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        PrintWriter out = resp.getWriter();
        if(!req.getParameter("display").equals("show")) {
            out.println("<form method="post" action="servlet/Test">");
            String[] result = req.getParameter("parameters").split(":");
            if(!req.getParameter("display").equals("show")) {
                out.println("<textarea name="parameters"/>");
                for (int x=0; x<result.length; x++) {
                    out.print(result[x]);
                    if(!req.getParameter("display").equals("show")) out.print(":");
                }
                if(!req.getParameter("display").equals("show")) { out.println("</textarea>");
                out.println("<input type="submit" value="ok"/>");
                out.println("</form>"};
            }
        }
    }
}
Method calls

Moving further along the “dynamic functionality” axis:

- method calls.

Different possibilities:

- REST / XML-over-HTTP;
- SOAP requests and SOAP responses;
- XML-RPC.
REST / XML-over-HTTP

The XML-over-HTTP “protocol”:

▶ send a request via HTTP;
▶ receive an XML document as response;
▶ extremely light-weight;

Example:

▶ Amazon

Recall our second definition of web service:

At a minimum, however, a Web service is any piece of software that makes itself available over the Internet and uses a standardized XML messaging system.

[“Top Ten FAQs for Web services”, 2001]
Example: Joke Server

The Joke Server we developed used XML-over-HTTP:

- the “list categories” operation required no parameters and returned an XML doc;
- the “get jokes” operation required a category parameter and returned an XML doc;
- the “submit joke” operation required a parameter in the body of the request, and returned an empty XML doc.

URL names ressources and (XML) representations are returned.

Key API routines:

- the HttpServlet.doPost callback;
- the HttpServletRequest.getParameter method;
- the HttpServletResponse.getWriter method;
- the HttpURLConnection.getOutputStream method.
XML-over-HTTP does not specify a standard format for message exchange — SOAP does.

- two message exchange styles:
  - document-style (one-way XML messages);
  - RPC (Remote Procedure Call, request-response).

- different protocol transports:
  - HTTP
  - SMTP
  - ...

SOAP messages:

- envelopes containing:
  - header (info for intermediaries);
  - body (actual message contents).
Recall the most restricted definition of a web service:

A Web service is a **software system** designed to support interoperable **machine-to-machine interaction** over a network. It has an **interface described** in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using **SOAP-messages**, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

[“Web Services Glossary”, 2004]
Example: SOAP enabled Joke Server

Could have used SOAP for the Joke Server:

▶ “list categories”: “empty” request, document-style response;
▶ “get jokes”: SOAP encoded request, document-style response;

Key technology:

▶ Axis (a SOAP engine).

Why Java? Why XML?

▶ Java: tool support.
▶ XML: obvious.
▶ however, many other possibilities (.NET, for example).
WSDL

How do we know how to talk to a service?

.specifying the interface to the service

```xml
<?xml version="1.0"?>
<definitions ...>
  <types> <!-- types used in messages --></types>
  <message> <!-- messages used in interactions --></message>
  <portType> <!-- interaction patterns --></portType>
  <binding> <!-- messaging frameworks for ops --></binding>
  <service> <!-- service address --></service>
</definitions>
```
Example: Specifying Joke Server interface in WSDL

Could have specified the Joke Server interface in WSDL.

▶ Did almost this in Axis lecture.

Key technology:

▶ Axis (more than just a SOAP engine)
▶ WSDL2Java (generate stub code for calling service)
▶ Java2WSDL (generate description from interface)

Why Java? Why XML?

▶ Java: tool support.
▶ XML: obvious.
▶ however, many other possibilities (.NET, for example).
How do we find the service in the first place?

- use service catalogs;
- service discovery and registration;
- UDDI, for example.
Example: Joke meta service

The Joke meta service is an example of a service catalog.
- supports registration of joke servers;
- allows lookup of servers.

In this case: Key technology:
- servlet with XML-over-HTTP.

More generally: Key technology:
- UDDI4J (Java class library);
- programmatic access to UDDI registries.

Why Java? Why XML?
- Java: tool support.
- XML: obvious.
- however, many other possibilities (.NET, for example).
Important subjects we haven’t dealt with

- concurrency issues and data consistency;
- security;
- design;
- application development frameworks;
Summary

Gone from

- single user applications (before the course),
- over web applications,
- to web services.
## Numerical scores (course)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>I am happy about this course</td>
<td>4.94</td>
</tr>
<tr>
<td>There is clear progression</td>
<td>5.00</td>
</tr>
<tr>
<td>Highly relevant for job profile</td>
<td>5.50</td>
</tr>
<tr>
<td>Contents is very pratical</td>
<td>5.17</td>
</tr>
<tr>
<td>I would like more emphasis on practice</td>
<td>3.78</td>
</tr>
<tr>
<td>Theoretical level is high</td>
<td>4.28</td>
</tr>
<tr>
<td>I would like the theo. level increased</td>
<td>4.28</td>
</tr>
<tr>
<td>Good knowledge of relevant litt.</td>
<td>4.33</td>
</tr>
<tr>
<td>I satisfy the course requirements</td>
<td>5.28</td>
</tr>
<tr>
<td>I spend a great deal of time (cmp. 15h)</td>
<td>3.67</td>
</tr>
<tr>
<td>I think a lot is expected (cmp. 15h)</td>
<td>3.22</td>
</tr>
</tbody>
</table>

(18 respondents)
I am happy about this course

Average: 4.96 (18 respondents)

“agree ++”: 7 (39%)
“agree +”: 7 (33%)
“agree”: 3 (17%)
“disagree”**: 1 (6%)
“disagree --”: 0 (0%)
“disagree -”: 1 (6%)

Henning Niss (ITU)
Numerical scores (tcd)

This is a teacher I am happy to have 5.12
Teacher uses good examples 4.94
Teacher gives a good subject overview 4.94
Teacher makes good use of practice 4.38
Teacher shows deep theoretical insight 4.69
Teacher provides good help and feedback 4.88
Teacher responds constructively 5.12
Teacher is open to new ideas 4.81
Teacher is enthusiastic 4.81
Teacher is always well-prepared 4.25
(16 respondents)
This is a teacher I am happy to have (tcd)

Average: 5.12 (16 respondents)

“I am happy to have this teacher”
16 respondents

“agree ++”: 7 (44%)
“agree +”: 6 (38%)
“agree”: 1 (6%)
“disagree”: 2 (13%)
“disagree -”: 0 (0%)
“disagree --”: 0 (0%)
Numerical scores (hniss)

This is a teacher I am happy to have 5.13
Teacher uses good examples 5.20
Teacher gives a good subject overview 4.67
Teacher makes good use of practice 4.67
Teacher shows deep theoretical insight 5.13
Teacher provides good help and feedback 4.60
Teacher responds constructively 4.60
Teacher is open to new ideas 4.80
Teacher is enthusiastic 4.80
Teacher is always well-prepared 5.00

(15 respondents)
This is a teacher I am happy to have (hniss)

Average: 5.13 (15 respondents)

“I am happy to have this teacher”
15 respondents

“agree ++”: 5 (33%)

“agree +”: 7 (47%)

“agree”: 3 (20%)

“disagree --”: 0 (0%)
“disagree -”: 0 (0%)
“disagree”: 0 (0%)
Overall conclusion: the good

- good course;
- practical and relevant;
- good overview;
- real-world mini projects.
Overall conclusion: the bad

- bad book
  - *should be published by next fall*;
- problems with tools
  - *more HOWTOs*;
- JWIG and Xact irrelevant
  - *highlights limitations and future ahead!*
- (exercises and exercises classes do not work)
  - *suggested solutions*;
  - *more manning*. 
“Based on your comments and scores, it seems that we have succeeded in giving you an overview of the area of web applications and services and the principles underlying them.”